CS 140 Project 4: File Systems

February 28, 2020

Today’s Topics

- **Overview**
- **Project 4 Requirements**
    - Buffer Cache
    - Indexed and Extensible Files
    - Subdirectories
    - Synchronization
- **Getting Started**

Project Overview

- **Build on top of project 2 or project 3**
  - Up to 5% extra credit if you enable VM
  - Edit ‘`filesys/Make.vars`’ to enable VM
- **Remove the severe limitations of the basic file system**
  - No internal synchronization
  - File size is fixed at creation time
  - File data is allocated on contiguous range of disk sectors
  - No subdirectory

Project Overview

Reference Implementation:

```
Makefile.build      |   5
devices/timer.c     |  42 ++
filesys/Make.vars   |   6
filesys/cache.c     | 473 +++++++++++++++++++++++++
filesys/cache.h     |  23 +
filesys/directory.c |  99 ++++-
filesys/directory.h |   3
filesys/file.c      |   4
filesys/filesys.c   | 194 +++++++++-
filesys/filesys.h   |   5
filesys/free-map.c  |  45 +-
filesys/free-map.h  |   4
filesys/fsutil.c    |   8
filesys/inode.c     | 444 ++++++++++++++++++-----
filesys/inode.h     |  11
... snip ...
```

Today’s Topics

- **Overview**
- **Project 4 Requirements**
  - Buffer Cache
  - Indexed and Extensible Files
  - Subdirectories
  - Synchronization
- **Getting Started**

Buffer Cache

- **Modify the file system to keep a cache of file blocks**
    - Reduce expensive disk I/O
    - No more than 64 sectors (including inode and file data)!
- **Get rid of the “bounce buffer” in** `inode_{read,write}_at()`
    - Used to implement read/write in byte-granularity
    - Interact with the buffer cache instead
- **Cache replacement algorithm**
    - Must be at least as good as the “clock” algorithm
    - Maybe give higher priorities to metadata (i.e., inode) over file data?

Buffer Cache, Cont’d

- **Your cache should be *write-behind***
  - Keep dirty blocks in cache
  - Write to disk on cache eviction
  - Periodically flush dirty blocks back to disk
  - Don’t forget to flush when Pintos halts (in `filesys_done()`)
- **Your cache should also be *read-ahead***
  - Prefetch the next block of a file when one block of file is read
  - Only meaningful when done asynchronously, in the background

Remove inode_disk from inode

```
/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start;   /* First data sector. */
    off_t length;           /* File size in bytes. */
    unsigned magic;         /* Magic number. */
    uint32_t unused[125];   /* Not used. */
  };

/* In-memory inode. */
struct inode
  {
    ... unrelated fields omitted ...
    ⬇ YOU SHOULD REMOVE THIS FIELD
    struct inode_disk data; /* Inode content. */
  };
```

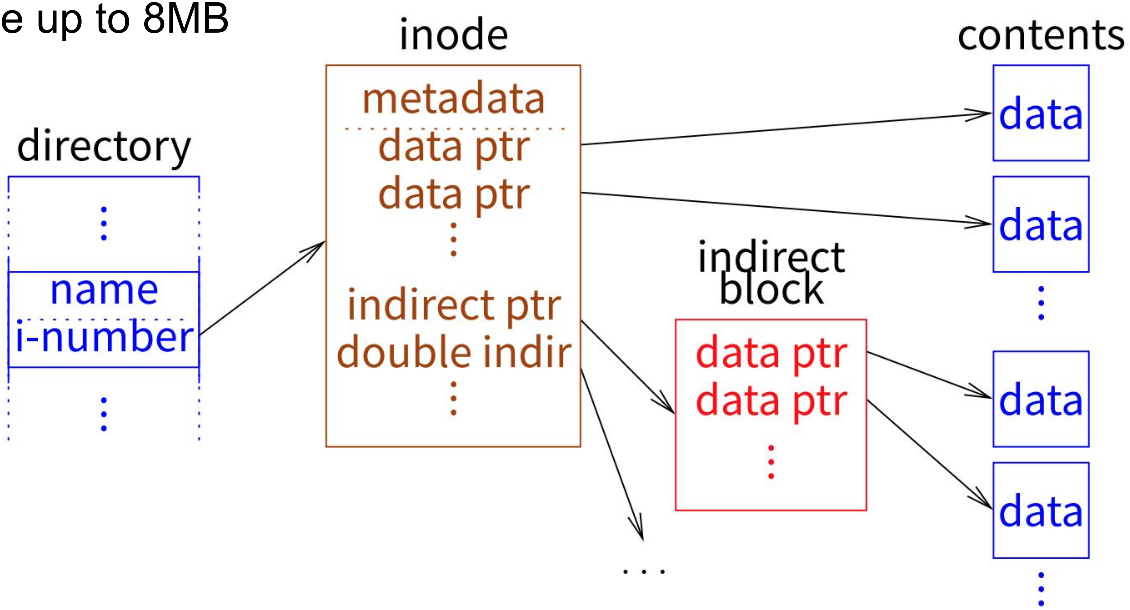
Indexed and Extensible Files

- **The basic file system suffers from external fragmentation**
  - Always allocates files as a single extent
  - Dictated by the current representation of an inode

```
/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start;   /* First data sector. */
    off_t length;           /* File size in bytes. */
    unsigned magic;         /* Magic number. */
    uint32_t unused[125];   /* Not used. */
  };
```

Indexed and Extensible Files, Cont’d

- **Modify `struct inode_disk` to use an index structure**
  - Use a combination of direct, indirect, and doubly indirect blocks
  - Support file size up to 8MB



Indexed and Extensible Files, Cont’d

- **Support file growth**
  - There should be no predetermined limit on the size of a file
  - File size starts as 0; expanded every time user writes beyond EOF
  - Details in Section 5.3.2
- **Directory can grow too: remove the 16-file limit in the root directory**
  - “`dir_create(ROOT_DIR_SECTOR, 16)`” in `filesys.c:do_format(void)`
- **Use the “free map” (`free-map.c`) to keep track of free disk sectors**
  - Hard-coded to be kept at disk sector 0 (i.e., “`#define FREE_MAP_SECTOR 0`”)
  - Note: You can keep a cached copy permanently in memory

Subdirectories

- **Implement a hierarchical name space**
  - E.g., “`/foo/bar/../baz/./a`”
  - Directory entries (i.e., `struct dir_entry`) can point to files or other directories
- **Each process has its own current directory**
  - Set to the root directory at startup
  - Inherited by the child process started by the **`exec`** system call
- **Implement path resolution**
  - Update existing syscalls to take path names (absolute or relative) as inputs
  - Support special file names ‘.’ and ‘..’

Subdirectories, Cont’d

- **Update existing system calls**
  - Update **`open`** to open directories
  - Update **`remove`** to delete empty directories
  - …
  - Many more details in Section 5.3.3
- **More system calls**
  - Implement `chdir, mkdir, readdir, isdir, and inumber`
  - User programs `ls`, `mkdir`, and `pwd` should work now

Synchronization

- **No more global file system lock**
  - Operations on different buffer cache blocks must be independent
  - E.g., process A can read cache block 3 while process B is replacing block 7
- **Multiple processes must be able to access the same file concurrently**
  - When the file size is fixed: read can see partial change; writes can interleave
  - But extending a file and writing data into the new section must be atomic
- **Operations on the same directory must be serialized**
  - Operations on different directories are independent

Today’s Topics

- **Overview**
- **Project 4 Requirements**
  - Buffer Cache
  - Indexed and Extensible Files
  - Subdirectories
  - Synchronization
- **Getting Started**

Getting Started

- **New code to work with**
  - `directory.h/c`: Performs directory operations using inodes
  - `inode.h/c`: Data structures representing the layout of a file's data on disk
  - `file.h/c`: Translates file reads and writes to disk sector reads and writes
  - Details in Section 5.1.1
- **Testing file system persistence**
  - Invoke Pintos a second time to copy files out of the Pintos file system
  - Grading scripts check if the contents of the file meet expectation
  - Won't pass the extended file system tests until you support `tar`
  - Details in Section 5.1.2

Suggested Order of Implementation

- **Buffer cache**
  - All tests from project 2 (or project 3) should still pass
- **Extensible files**
  - Pass the file growth tests
- **Subdirectories**
  - Pass the directory tests
  - Can be done more or less in parallel with extensible files

Think about synchronization from the beginning.

Questions?